Spécification détaillée du produit SimPA2

France, USA, Germany, Japan, Korea, United Kingdom, Brazil, Hungary

IMAGINE SA - 7, place des Minimes – 42300 Roanne FRANCE

Phone +33 4 77 23 60 30 - Fax +33 4 77 23 60 31

E-mail : imagine@amesim.com

www.amesim.com

Advanced Tools & Expertise in Systems Engineering
Aeronautics - Automotive - Power Hydraulics

Contributors:

Contributors	Company	Function	Email	Telephone
D. TALBI	LMS Imagine	Developer	talbi@amesim.com	+(33)477236030
S. FURIC	LMS Imagine	Development Project Manager	furic@amesim.com	+(33)477236030
D. FARGETON	LMS Imagine	Product Director	fargeton@amesim.com	
M.Najafi	INRIA	Ingénieur Expert	masoud.najafi@inria.fr	+(33)139635907
N. CUVILLIER	ALYOTECH (CRIL)	Developer	nicolas.cuvillier@alyotech.fr	+(33)130672349

Document revisions:

Version No.	Date	Author	Observations
1	November 13th 2007	D. TALBI	Initial revision
1	November 16th 2007	S. FURIC	
1.1	November 19th 2007	D. FARGETON	Adjustments before send to INRIA & CRIL
1.2	November 30th 2007	N. CUVILLIER	Adjustments by ALYOTECH (CRIL) See § 5.3, 6.3.7, 6.4.1, 6.4.2, 6.7.6, 6.8.2
1.3	Novembre 30th 2007	M. Najafi	partie Scicos
1.4	December 3d 2007	D. FARGETON	Adjustments before first official delivery
1.5	December 4th 2007	D. TALBI	Remove revision for official delivery
1.6	January 23rd 2008	D. TALBI	Update

Checked:

Who	Company	Function	Date / Signature
S.Furic	LMS Imagine	Development Project Manager	Dec 4th 2007

Approved:

Who	Company	Function	Date / Signature
D.Fargeton	LMS Imagine	Product Director	Dec 4th 2007

Table of contents

1 Glossaire

Caml ou Ocaml: langage de programmation développé et distribué par l'INRIA depuis 1985 (consulter le site http://caml.inria.fr pour plus de détails).

Modelica: langage pour la modélisation des systèmes physiques, développé par l'association Modelica (consulter le site http://www.Modelica.org/ pour plus de détails).

Classe: fait référence au concept de « classe » du langage Modelica. Une classe fournit la description des objets qui lui appartiennent (cf. [7]).

Equation: fait référence au concept d'équation dans le langage Modelica (cf. [5] Chapter 8).

Expression: fait référence à la notion d'expression telle que décrite dans la grammaire du langage Modelica (cf. [5] Appendix B).

Graphe: est un ensemble de sommets (appelés aussi nœuds) reliés par des arcs (cas d'un graphe orienté), ou des arêtes (cas d'un graphe non orienté).

IHM (Interface Homme-Machine): dénote l'ensemble des moyens mis en place pour assurer la communication entre l'homme et la machine.

Instance: un objet particulier en tant qu'élément d'une classe donnée.

Instanciation: procédure permettant d'obtenir un objet avec un paramétrage donné à partir d'une classe du langage Modelica.

Objet: entité du langage représentant :

- les modèles manipulés par l'utilisateur,
- les données structurées (enregistrements).

Type: cf. [5] Chapter 6, pour une définition de ce concept.

Variabilité: concept du langage Modelica, permettant de contraindre l'ensemble des valeurs pouvant être prises au cours du temps, par une expression.

XML: « eXtensible Markup Language » est un langage utilisé principalement pour le stockage et l'échange de données.

Modelicac : Ancienne version du compilateur Modelica, développée au cours du projet SimPA.

2 Introduction

SimPA2 (Simulation pour le Procédé et l'Automatique) est un projet labellisé ANR/RNTL en 2005, dont l'objectif principal est le développement d'un compilateur Modelica open source et son intégration au produit open source Scilab/Scicos et au produit commercial AMESim. Ce projet fait suite au projet SimPA qui a permis d'obtenir une première version de compilateur Modelica open source, qui sera utilisée comme point de départ pour l'implémentation du compilateur Modelica de SimPA2.

Ce document est la spécification détaillée du projet SimPA2 requise dans le cadre du lot 0 du sous-projet 1 (cf. l'annexe technique du projet [2]). Il est divisé en quatre parties :

- Dans les trois premières parties, nous donnerons une description du point de vue fonctionnel et architecture logicielle du compilateur Modelica, de l'intégration de Modelica dans l'environnement commercial AMESim, ainsi que son intégration dans l'environnement open source Scilab/Scicos.
- La quatrième partie est une réponse à la liste des exigences fonctionnelles [3] établie par les partenaires industriels (cf. l'annexe technique du projet [2]).

Dans ce qui suit, nous appellerons compilateur de SimPA l'ancienne version du compilateur développée au cours du projet SimPA (appelée aussi Modelicac). La nouvelle version (celle du projet SimPA2, dont ce document est la spécification détaillée) sera dénommée « compilateur Modelica ».

3 Compilateur Modelica

Le compilateur Modelica qui sera décrit ici est un produit open source entièrement développé dans le langage Objective Caml. Il permet d'analyser du code Modelica, de l'interpréter et de générer en sortie du code à destination d'un outil de simulation de systèmes physiques. Le compilateur reçoit en entrée un ensemble de fichiers contenant du code Modelica sous l'une des deux formes (cf. figure 1) :

- textuel (fichiers avec l'extension « .mo »)
- crypté en binaire (fichiers avec l'extension « .moc »).

Ce code Modelica est traité en plusieurs étapes :

- Analyse lexicale et grammaticale,
- Analyse sémantique,
- Manipulations symboliques,
- Génération de code pour un simulateur donné.

Ces quatre étapes seront décrites dans les parties 3.1 à 3.4.

Plusieurs types de codes peuvent être générés par le compilateur :

- du code XML contenant des informations syntaxiques et de type : en particulier, ce code contient les données stockées dans les annotations Modelica (informations graphique, documentation, « versionning »…)
- du code ciblé à destination d'une plate-forme de simulation donnée (ex. le logiciel AMESim ou Scicos) : ce type de code contient le code à utiliser pour la simulation (ex. code C pour le logiciel AMESim ou Scicos), ainsi que des informations sur la structure des instances à simuler (ex. description XML d'un sous modèle AMESim)
- un fichier binaire (.moc) contenant une description des systèmes à traiter, obtenue suite aux étapes d'analyse lexicale, grammaticale et sémantique : l'utilité de ce cryptage binaire est d'assurer plus de sécurité dans le stockage des modèles, et aussi de réduire le temps de compilation en effectuant certains traitements une fois pour toutes avant de générer le fichier

crypté ; ainsi il est possible d'éviter de refaire les vérifications de syntaxe et de type pour certaines librairies, en utilisant des fichiers cryptés à la place des fichiers textes.

Plusieurs options de traitement peuvent être transmises en argument du compilateur :

- « -no-simplifs » : permet d'interdire les simplifications de variables au cours de l'étape de manipulations symboliques (toutes les variables et paramètres du système initial sont conservés),
- « -jac » : permet de générer la matrice jacobienne du système sous forme analytique,
- « -c » : permet d'effectuer uniquement les analyses lexicale, grammaticale et sémantique, et génère ensuite un fichier crypté (.moc),
- « -lib » : permet d'indiquer l'emplacement d'un fichier Modelica à ajouter à la liste des fichiers à traiter,
- « -command » : transmet au compilateur la commande à exécuter pour instancier le modèle à simuler.

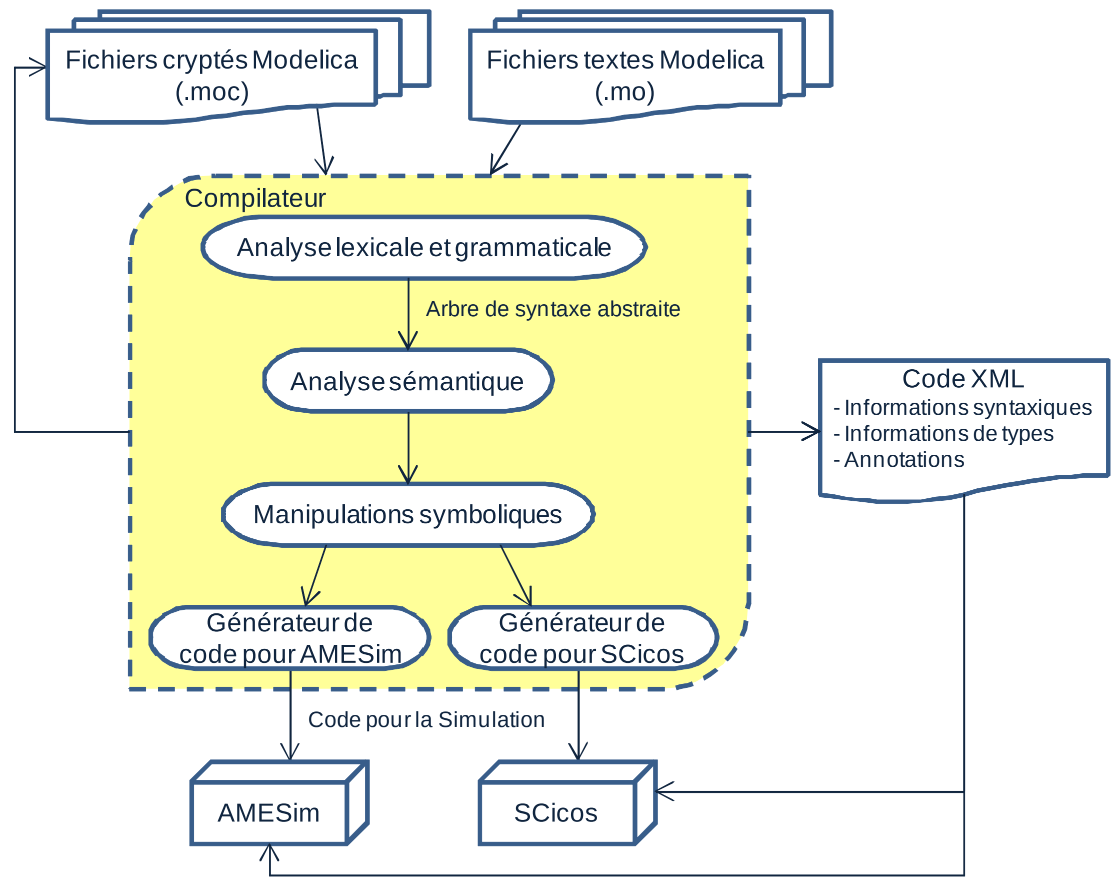


Figure 1 – *Schéma d'utilisation du compilateur Modelica*

3.1 Analyse lexicale et grammaticale

Au cours de l'analyse lexicale le flux de caractères (provenant du code source Modelica) est transformé en une suite de lexèmes. Chaque lexème représente une entité élémentaire du langage Modelica : mot-clé, opérateur, identifiant, commentaire… Si des caractères non autorisés sont détectés, une erreur lexicale est affichée.

Les lexèmes sont ensuite regroupés en phrases par l'analyseur grammatical qui détecte les phrases mal formées et renvoie des messages d'erreurs de syntaxes éventuels. Si cette phase est passée avec succès, un arbre de syntaxe abstraite est produit. Celui-ci est conforme à la syntaxe décrite dans la spécification 3.0 du langage Modelica [5], avec une extension de la syntaxe des expressions. Celle-ci a été rajoutée afin

de respecter les mêmes règles standards utilisées par les autres langages de programmation, comme C ou Java.

Si une erreur se produit au cours de cette étape d'analyse, une erreur de syntaxe est affichée avec les détails sur l'emplacement du fichier source, ainsi que la position de l'erreur (ligne et colonne).

3.2 Analyse sémantique

Au cours de cette étape, le compilateur tente de donner des sens aux phrases (grammaticalement correctes) obtenues suite à l'étape précédente (analyse lexicale et grammaticale). Si une phrase n'a pas de sens ou introduit une construction du langage non supportée par le compilateur, une erreur de sémantique est affichée, avec des indications sur l'origine et l'emplacement, ainsi que la position de l'erreur. Le sous-ensemble non supporté du langage Modelica fera l'objet d'un autre document.

Au cours de l'analyse sémantique, plusieurs traitements sont effectués afin de réécrire le système à traiter sous une forme interne adaptée, avant d'être transmis à l'étape de manipulations symboliques. Parmi les traitements qui sont effectués :

- mise à plat des équations et variables multidimensionnelles,
- transformation des équations conditionnelles en expressions conditionnelles,
- traitement des conversions implicites de types (ex. conversions d'entiers en réels),
- réécriture d'une partie des opérateurs et fonctions internes au langage « built-in »,
- génération des équations de connexion,
- substitution des paramètres fixés (i.e. attribut « fixed » égal à « true ») par leurs valeurs dans les équations.

Si une erreur se produit au cours de cette étape d'analyse, une erreur de sémantique est affichée avec les détails sur l'emplacement du fichier source, ainsi que la position de l'erreur (ligne et colonne).

3.3 Manipulations symboliques

Les manipulations symboliques couvrent l'analyse structurelle du modèle ainsi que les opérations de transformation appliquées pour générer un code exploitable par un environnement de simulation. La représentation interne reçue de l'étape précédente (analyse sémantique) est exploitée pour construire deux systèmes de contraintes : continu et discret.

La représentation interne utilisée en entrée comprend l'ensemble des paramètres « non fixés » et variables du modèle à analyser, les valeurs initiales ainsi que les équations.

Le système de contraintes continues est décomposé en sous-systèmes autonomes par un module d'analyse de graphe d'incidence. Ces sous systèmes sont réduits par manipulations symboliques, en résolvant les parties linéaires et en réduisant au mieux les parties non-linéaires (par des méthodes d'inversion et de substitution symboliques). Le système simplifié obtenu à la fin, est constitué de sous-systèmes non-réductibles qui devront être traités numériquement. Le résultat comprendra éventuellement les paramètres dont on ne connaît pas la valeur.

Les systèmes discrets sont ordonnancés statiquement par une méthode basée sur un algorithme issu du compilateur Scicos, fourni par l'INRIA. Les équations considérées ici seront « explicites », au sens où on va exiger que le membre de gauche soit la variable par rapport à laquelle on va résoudre l'équation en question.

3.4 Génération de code ciblé pour un simulateur donné

La génération de code permet aux applications clientes de récupérer de l'information structurelle sur les modèles à analyser ou du code C compilable dont l'exécution dans un environnement de simulation permet d'obtenir les courbes d'évolution temporelles d'un système dynamique hybride.

Deux cibles pour la génération de code ont été définies :

- Scicos
- AMESim

La génération vers la première cible consiste à produire un fichier C contenant le code d'un bloc Scicos comportant des sections de calcul de résidus, de spécification de valeurs initiales, de détection de traversées de surface, de calcul de coefficients de matrice Jacobienne… Pour les besoins d'analyse, un fichier XML donnant la structure du système d'équations pourra être généré.

La génération vers AMESim consiste à produire deux fichiers :

- Un fichier C décrivant le code du sous-modèle AMESim correspondant au modèle Modelica compilé
- Un fichier XML décrivant l'interface du sous-modèle.

4 Intégration de Modelica dans AMESim

4.1 Simulation directe sous AMESim version Rev7A

La version Rev7A d'AMESim (sortie en mai 2007) inclut un assistant d'import Modelica. Cet assistant demande à l'utilisateur d'indiquer (cf. figure 2) :

- les chemins vers les sources Modelica à compiler
- le nom de la classe à instancier
- le nom de l'instance à générer
- le répertoire qui sera utilisé pour stocker le code généré.

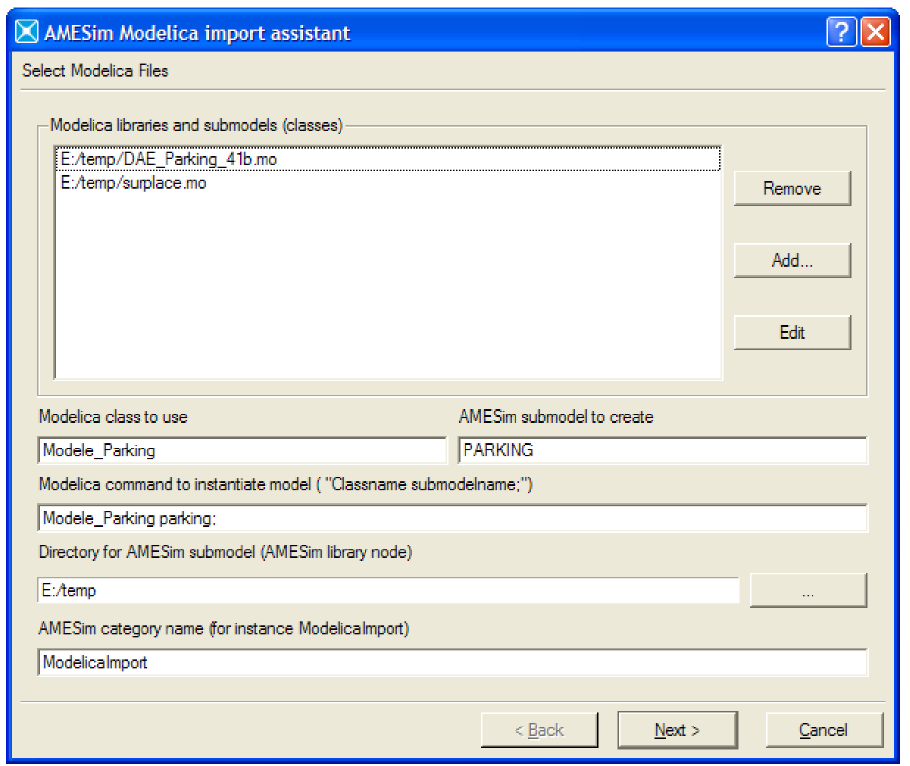


Figure 2 – *Configuration de l'assistant d'import Modelica.*

Suite à cette étape de configuration, l'assistant appelle le compilateur Modelica avec les arguments appropriés, et génère (si aucune erreur ne se produit) le code C et XML permettant de décrire un sous-modèle AMESim.

L'assistant peut générer deux types de sous-modèles AMESim :
- un sous-modèle autonome (sans ports)
- un sous-modèle connectable à des sous-modèles issus de la compilation de code Modelica ou de la bibliothèque standard AMESim (cf. figure 3).

Concernant le comportement des sous modèles deux options sont possibles :
- le sous-modèle expose toutes ses variables internes à l'utilisateur (mode par défaut)
- le sous-modèle ne transmet que ses variables de sortie (mode « boîte noire »).

Dans cette première version de l'import Modelica, les variables et équations discrètes ne sont pas supportées. Pour plus de détails sur l'utilisation de l'assistant d'import, un manuel d'utilisateur est mis à disposition avec l'aide en ligne d'AMESim.

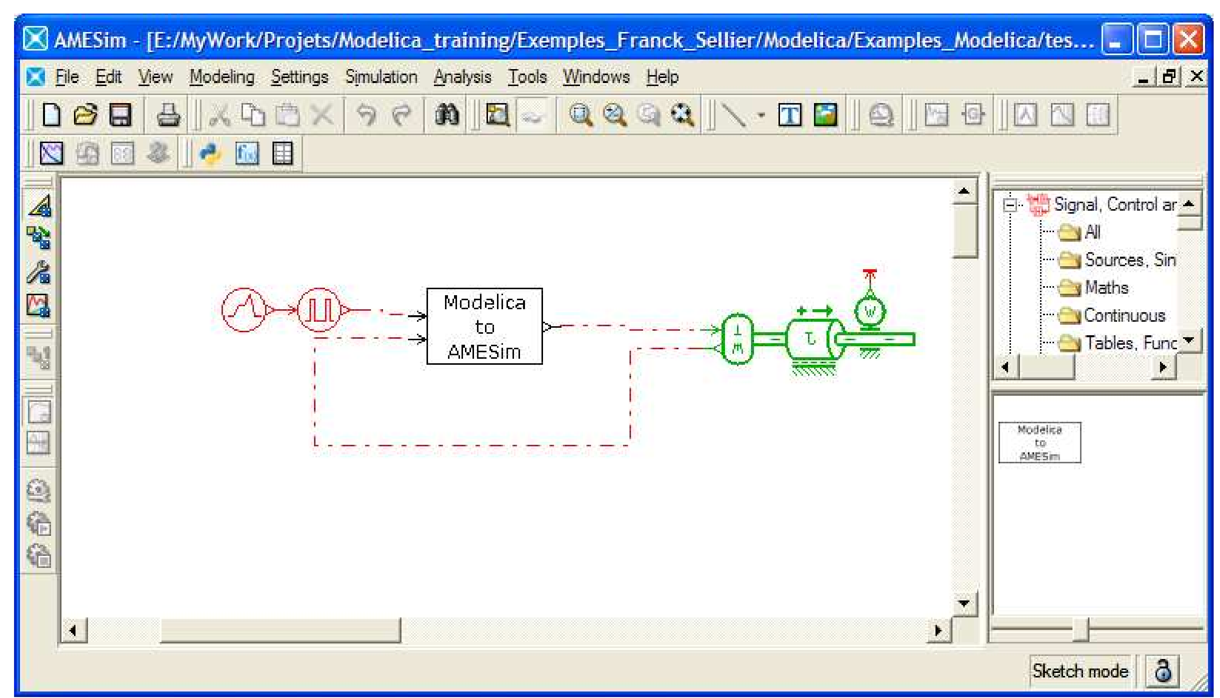


***Figure 3 –** Sous-modèle AMESim constitué d'un composant Modelica connecté à des composants AMESim classiques.*

4.2 Simulation directe sous AMESim version Rev8A

Parallèlement à la version Rev8A d'AMESim (prévue en mai 2008) une version dédiée à des projets de recherche offrira la possibilité de construire un modèle Modelica par assemblage graphique de composants (sans passer par la version textuelle). L'utilisateur pourra éditer un sous-modèle Modelica directement sous AMESim à l'aide de l'interface graphique. Le mélange de composants Modelica et AMESim ne sera pas autorisé, la communication entre les mondes Modelica et AMESim se réalisera à travers des ports (physiques ou signaux). La modélisation se fera donc en deux étapes :
- Construction d'un sous-modèle AMESim par assemblage de composants Modelica
- Incorporation du résultat à un modèle (circuit) AMESim et utilisation traditionnelle d'AMESim.

Les variables et équations discrètes ne seront pas supportées dans la partie Modelica.

4.3 Simulation directe sous AMESim version Rev9A

A partir de la version Rev9A d'AMESim (prévue en Mai 2009), les variables et équations discrètes seront supportées dans la partie Modelica.

4.4 Dimensionnement statique sous AMESim

Un prototype destiné au dimensionnement statique sous AMESim est disponible pour des projets de recherche (depuis mars 2007). Celui-ci offre la possibilité de modifier la nature des variables et paramètres, ainsi il est possible de :

- changer une variable en paramètre
- changer un paramètre en variable
- indiquer si un paramètre est fixé ou non (modification de l'attribut « fixed » en Modelica).

Ces modifications sont effectuées à partir d'une interface graphique qui affiche la liste des variables et paramètres du modèle (cf. figure 4), l'utilisateur peut alors indiquer :

- le type, en choisissant entre : variable, paramètre ou paramètre fixé
- la valeur initiale
- un commentaire.

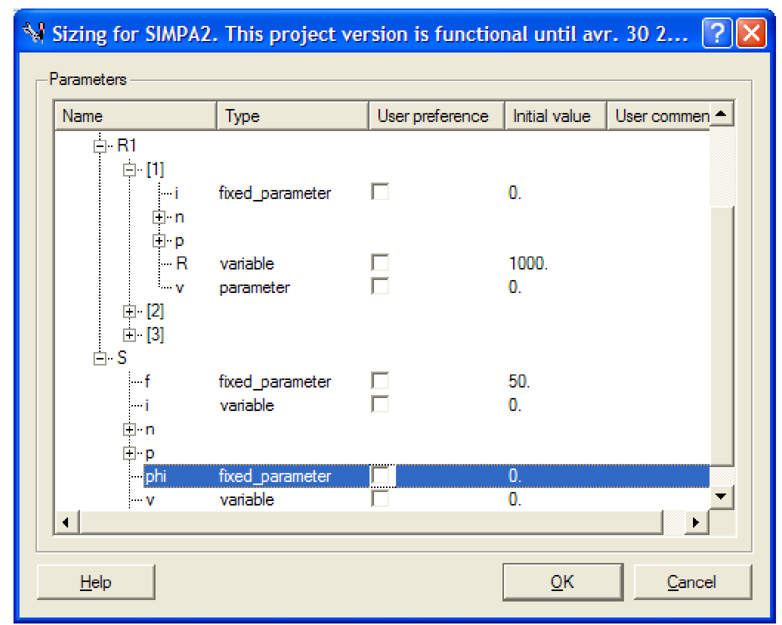


***Figure 4 –** Interface graphique du module de dimensionnement statique sous AMESim.*

Cette interface graphique utilise une description XML construite à partir du fichier source Modelica, un nouveau code Modelica est ensuite généré à partir de la description XML, en en tenant compte des choix effectués par l'utilisateur. Ce code Modelica est ensuite compilé pour générer un sous-modèle AMESim.

L'utilisateur se retrouve alors confronté à l'une des deux situations suivantes :

- la compilation s'est effectuée avec succès, dans ce cas un sous modèle AMESim est généré (code « C » et fichier de spécification « spe »),
- une erreur s'est produite au cours de la compilation. Ce qui indique que le système obtenu est structurellement singulier (e.g. trop ou peu d'équations). Dans ce cas l'utilisateur doit revenir sur ses choix, car il peut avoir fixé trop de variables, ou relâché trop de paramètres. Une approche automatique pour aider l'utilisateur dans son choix des variables à relâcher et à fixer est un sujet de recherche qui mérite d'être étudié par la suite (en dehors du projet SimPA2).

Après que la compilation du modèle modifié s'est effectuée avec succès, l'utilisateur peut lancer une ou plusieurs simulations et analyser les résultats obtenus. Il peut ensuite soit accepter les résultats de la simulation, ou bien reconsidérer ses choix, en fixant et en relâchant d'autres variables ou paramètres. La figure 5 ci-dessous décrit les différentes étapes d'utilisation du module de dimensionnement.

La première intégration effective du dimensionnement statique sous AMESim est prévue pour la version Rev 9A (mai 2009).

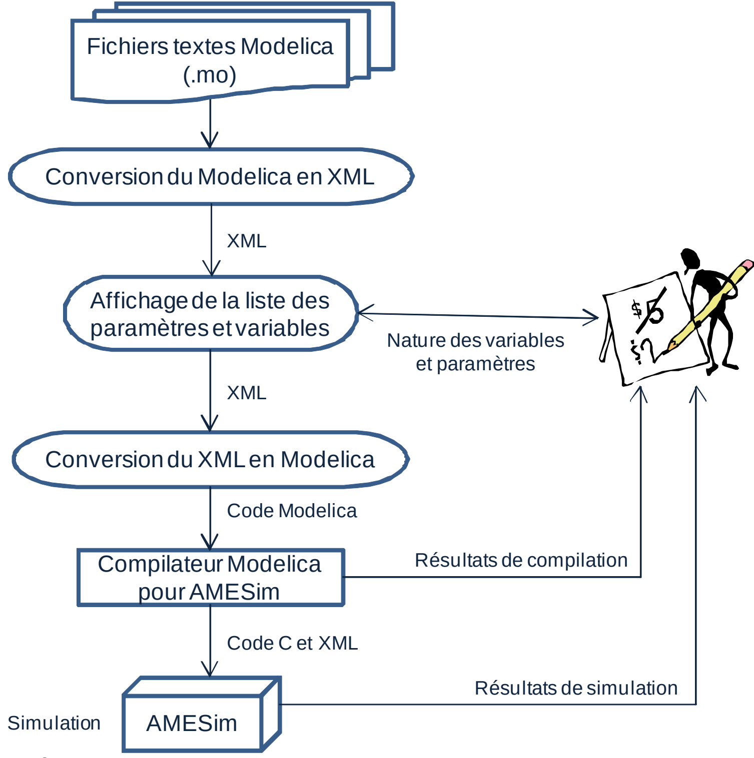


Figure 5 – *Schéma d'utilisation du module de dimensionnement statique sous AMESim.*

5 Intégration de Modelica dans Scicos

5.1 Simulation sous Scicos 4.2

La version 4.2 de Scicos utilise pour la compilation du code Modelica :
- une partie amont (appelée « Translator »),
- une partie aval (appelée « Modelicac »).

Modelicac sert à compiler les classes Modelica (fichiers avec l'extension « .mo ») et à générer des fichiers compilés (cryptés en binaire) avec l'extension « .moc ». Le deuxième rôle de Modelicac consiste à compiler le modèle complet et à générer un code pour une cible, en l'occurrence, Scicos. Pour effectuer cette seconde tâche, Modelicac a besoin d'utiliser les fichiers « .moc » déjà créés. Le chemin vers ces fichiers compilés est donné par la variable « modelica_libs». Par défaut cette variable contient les chemins vers les deux bibliothèques Electrical et Thermohydraulics de Scicos, ainsi que vers le répertoire temporaire des fichiers « .moc » généré au moment de la compilation des schémas Scicos.

Translator a pour rôle d'utiliser les bibliothèques Modelica existantes et de mettre à plat le Modèle Modelica complet.

La compilation des modèles Modelica passe par les étapes suivantes :
- écrire chaque programme Modelica dans un fichier séparé « .mo »
- générer des fichiers « .moc » pour chaque programme Modelica « .mo »
- compiler le programme Modelica complet « .mo » et générer le code C utilisable dans Scicos.

Pour pouvoir utiliser les modèles Modelica dans Scicos, chaque Modèle doit être associé à un fichier interface de Scicos « .sci ». Ce fichier définit les entrées/sorties implicites/explicites et le la forme graphique du bloc dans Scicos. La figure 6 montre le modèle d'un système de contrôle de moteur (développé par l'IFP) dans Scicos. Dans les diagrammes Scicos, l'utilisation simultanée de blocs Scicos standard et de blocs Modelica est autorisée.

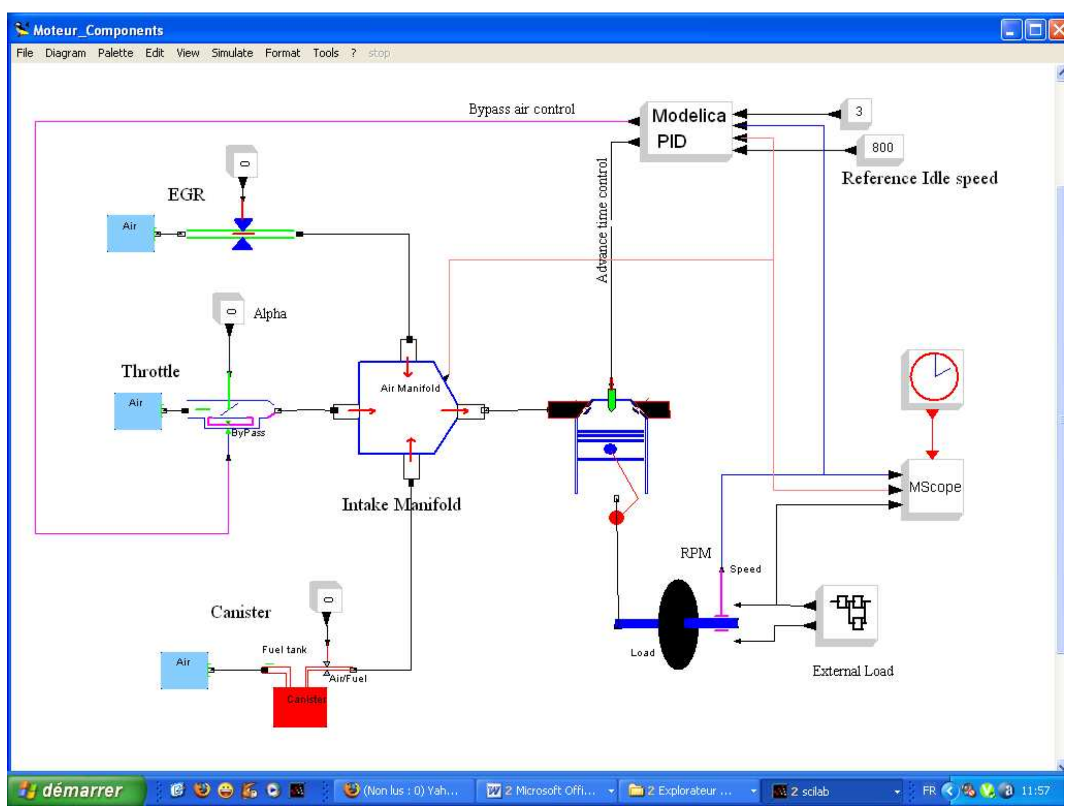


***Figure 6 –** Schéma d'utilisation des composants Modelica dans Scicos.*

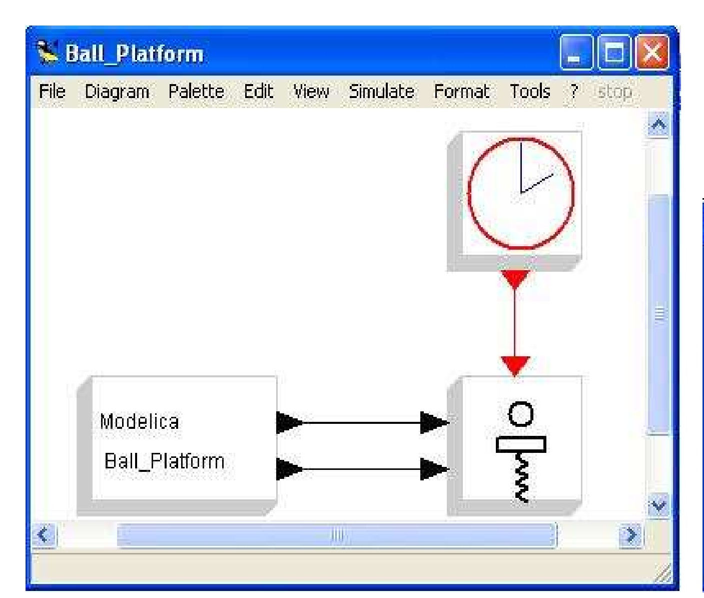
Il est également possible d'utiliser des modèles Modelica dans les diagrammes Scicos sans avoir à fournir un fichier d'interface, en utilisant une interface graphique générique pour les modèles Modelica (bloc « MBlock », cf. figure 7).

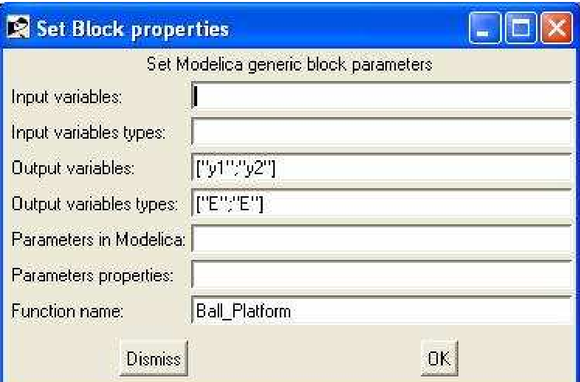
5.2 Dimensionnement statique sous Scicos

Le compilateur Modelica a été utilisé pour le développement d'une IHM permettant l'initialisation des modèles Modelica ainsi que le dimensionnement statique sous Scicos. Le dispositif d'initialisation/dimensionnement offre la possibilité de modifier la nature des variables et paramètres dans les modèles Modelica.

Ainsi il est possible de :
- libérer ou fixer une variable à une valeur définie,
- libérer ou fixer la dérivée d'une variable à une valeur définie,
- libérer ou fixer un paramètre à une valeur définie,
- associer un « confidence factor » aux variables générales (variables/paramètres/dérivées),
- associer des valeurs « max », « min » et « nominal » aux variables générales.

Le système d'équations ainsi obtenu ne sera pas nécessairement carré. Pour résoudre le système d'équations, Scicos fera appel à certains codes comme « optim », par exemple. La valeur des variables générales libérées est calculée puis utilisée dans la simulation dynamique du modèle Modelica.
L'organigramme d'initialisation actuel de Scicos est décrit dans la figure 8.





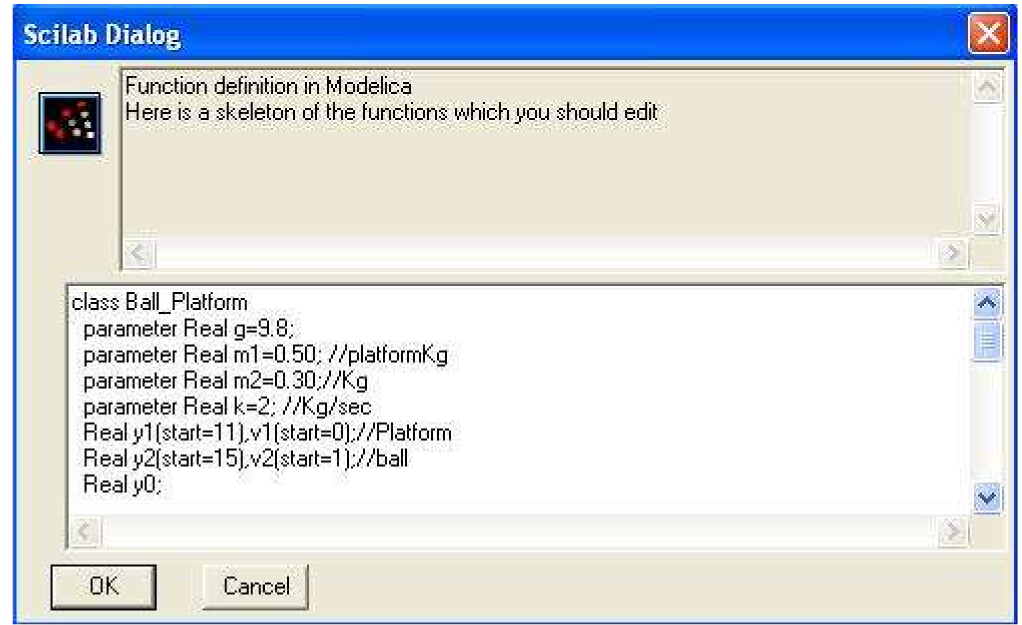


Figure 7 – Schéma d'utilisation du bloc MBlock dans Scicos.

Ces modifications sont effectuées à partir d'une interface graphique qui affiche la liste des variables générales du modèle (cf. figure 9). Cette interface graphique utilise une description XML construite à partir du fichier source Modelica.

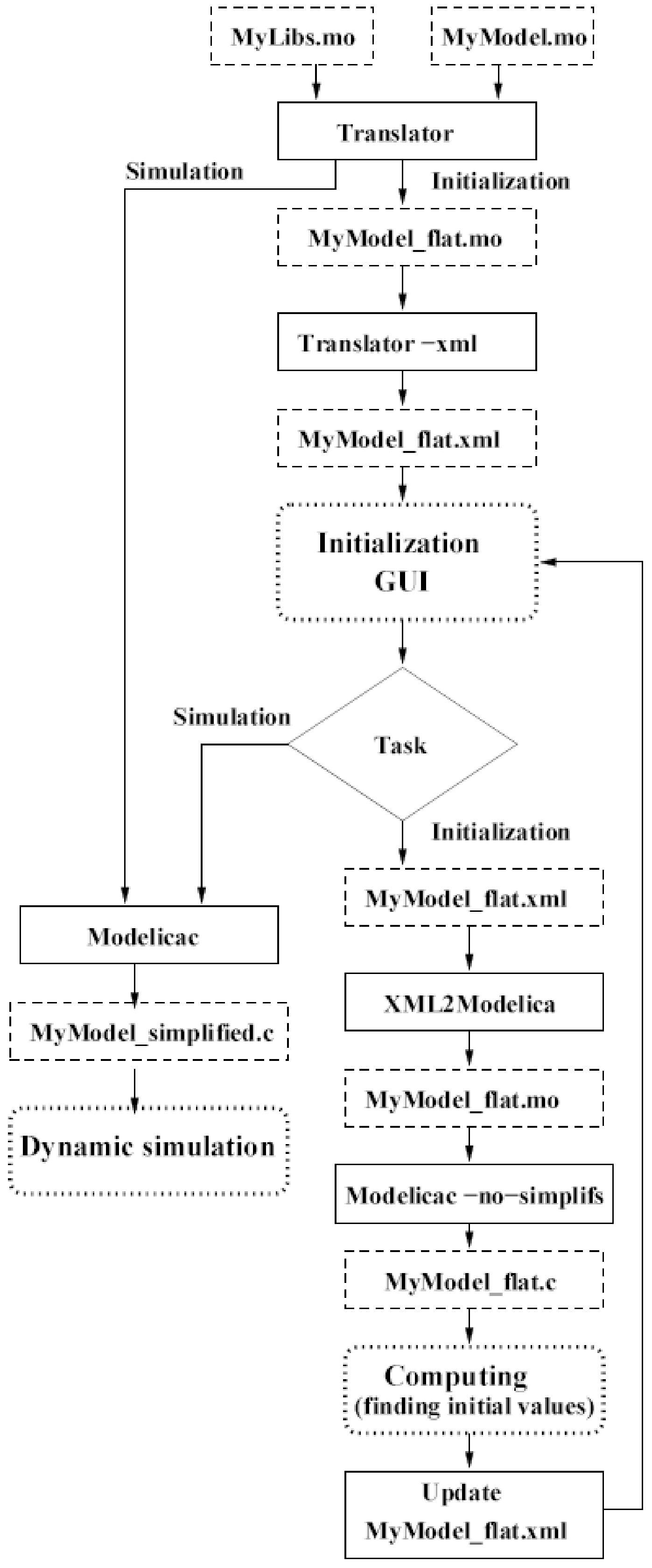


Figure 8 – *L'organigramme d'initialisation dans Scicos.*

Dans cette IHM, l'utilisateur peut alors modifier :
- la valeur initiale des variables générales,
- le poids pour indiquer le facteur de confiance (1 est identique au « fixed=true » de Modelica et une valeur entre 0 et 1 peut être attribuée au poids),
- un maximum,
- un minimum,
- une valeur nominale,
- un commentaire,
- une option pour filtrer et simplifier l'affichage des variables générales.

En plus l'interface graphique offre :
- une vue arborescente du modèle Modelica. En cliquant sur des nœuds de l'arbre, les variables générales associées aux nœuds sont affichées dans le tableau à droite
- la possibilité de chercher et afficher des variables, afficher les variables sélectionnées par nœud, afficher les variables sélectionnées à plat, afficher les variables dont l'attribut « fixed » est changé (par nœud, et à plat)
- la possibilité d'afficher plusieurs Modèles Modelica à la fois
- la possibilité de sauvegarder un fichier XML sous un autre nom pour le réutiliser plus tard (cette option est particulièrement utile quand l'utilisateur veut avoir plusieurs configurations pour un seul modèle Modelica)
- des informations sur le modèle comme le nombre de variables fixées, de variables libérées, de paramètres fixés, de paramètres libérés, ainsi que le nombre d'équations, peuvent être affichées.

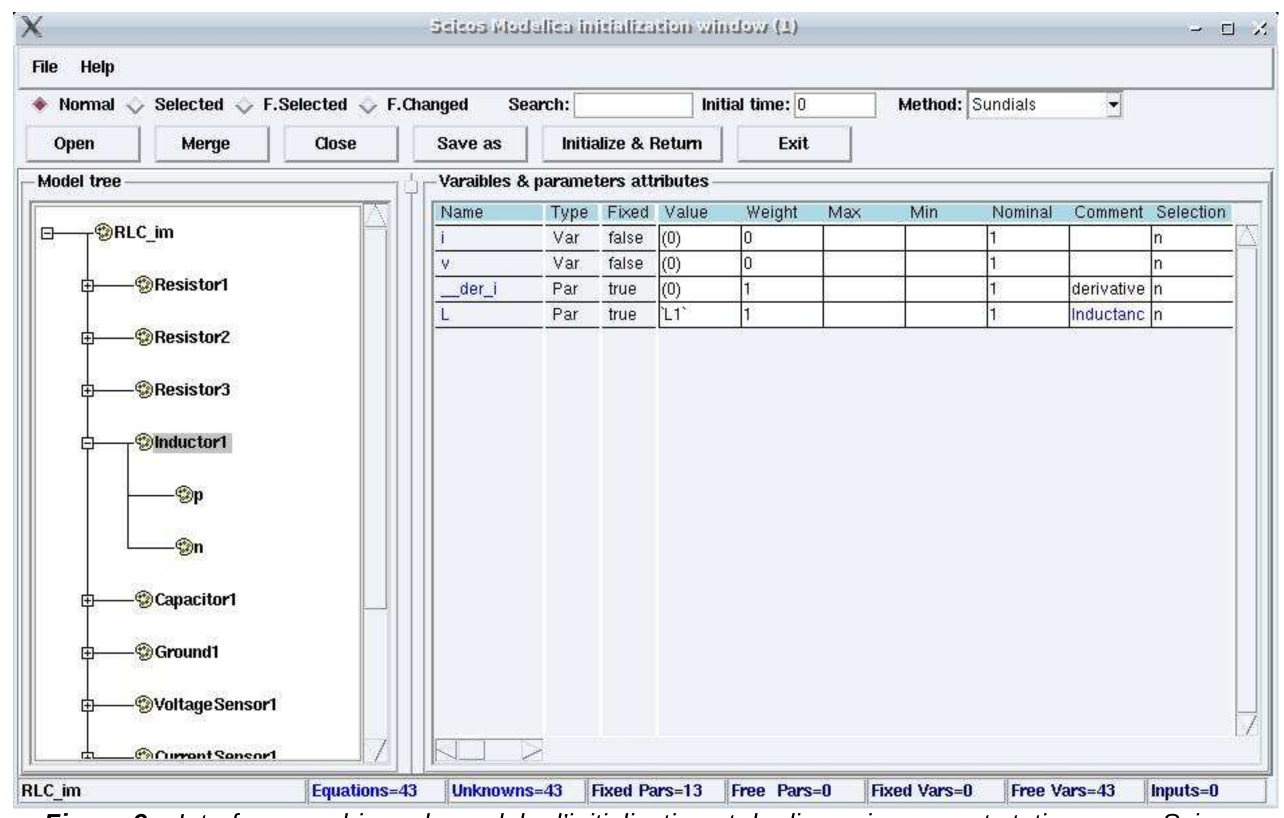


Figure 9 – *Interface graphique du module d'initialisation et de dimensionnement statique sous Scicos.*

L'IHM d'initialisation permet à l'utilisateur de choisir entre plusieurs méthodes d'initialisation :
- **Sundials** : qui utilise la méthode Newton modifiée.
- **Fsolve** : qui est le solveur interne de Scilab et utilise la méthode Newton avec une factorisation QR
- **Optim** : qui est un optimisateur interne de Scilab. ce solveur est notamment utilisé pour sortir des zones où la matrice Jacobian est singulière.

- **Homotopy** : qui utilise le « package » HOMPACK (http://www.netlib.org/hompack/) permettant la résolution d'équations par la méthode d'homotopie (cf. [8]).
- **Fsolve** : qui utilise aussi la méthode d'homotopie, mais avec le solveur Fsolve.

5.3 Problème inverse sous Scicos

L'objectif de cette partie est le support de l'inversion de modèles statiques et dynamiques pour la résolution de problèmes inverses (tels que la détermination de l'état initial de systèmes thermodynamiques ou le dimensionnement d'une chaîne d'actionnement de systèmes mécatroniques). La partie IHM, pour choisir les paramètres/variables, modifier leurs attributs, et fixer ou libérer leurs valeurs est pratiquement finie. Pour la partie simulation/initialisation, certains solveurs sont déjà implémentés et nous sommes en train d'enrichir et d'augmenter les performances des solveurs intégrés dans Scicos.

Ce qui est également important dans le dimensionnement statique et le dimensionnement inverse est de bien choisir des variables/paramètres à fixer ou à libérer. Cette méthodologie par dimensionnement inverse devrait mettre au point par le laboratoire AMPERE de l'INSA à Lyon.

5.3.1 Modèle de données

Le modèle de données repose sur les concepts physiques de base et les associations entre eux. Ce modèle a été traduit en langage informatique par l'intermédiaire d'un diagramme UML (voir figure 12).

Les concepts de base ainsi que les relations entre eux peuvent être décrits à partir de ce modèle UML. Ce modèle nous sert de base à la conception de l'application permettant de décrire les systèmes statiques ou dynamiques auxquels nous souhaitons appliquer la méthodologie de dimensionnement inverse.

Concepts	Données[1]	Contraintes	Commentaires
Concept ① Structure énergétique	n_{J1} J1 : $\{J1_1,\cdots,J1_{n_{J1}}\}$		Eléments de conservation d'énergie.
	n_{J0} J0 : $\{J0_1,\cdots,J0_{n_{J0}}\}$		
	n_{TF} TF : $\{TF_1,\cdots,TF_{n_{TF}}\}$		Eléments de transduction d'énergie conservatrice en puissance.
	n_{GY} GY : $\{GY_1,\cdots,GY_{n_{GY}}\}$		
Concept ② Puissances	n_P P : $\{P_1,\cdots,P_{n_P}\}$		Puissances.
Concept ③ Lois de conservation	n_{J1} LCB1 : $\{LCB1_1,\cdots,LCB1_{n_{J1}}\}$		Bilans de variables d'effort.
	n_{J0} LCB0 : $\{LCB0_1,\cdots,LCB0_{n_{J0}}\}$		Bilans de variables de flux.
	n_{LCE1} LCE1 : $\{LCE1_1,\cdots,LCE1_{n_{LCE1}}\}$		Egalités entre variables de flux.
	n_{LCE0} LCE0 : $\{LCE0_1,\cdots,LCE0_{n_{LCE0}}\}$		Egalités entre variables d'effort.
	$n_{J1}+n_{J0}$ BP : $\{BP_1,\cdots,BP_{n_{J1}},BP_{n_{J1}+1},\cdots,BP_{n_{J1}+n_{J0}}\}$		Bilans de puissance.
Concept ④ Lois de transduction d'énergie conservatrice	$2(n_{TF}+n_{GY})$ LTE : $\{LTE_1,\cdots,LTE_{2n_{TF}},LTE_{2n_{TF}+1},\cdots,LTE_{2(n_{TF}+n_{GY})}\}$		Lois de transduction d'energie conservatrice en puissance.
	$n_{TF}+n_{GY}$ EP : $\{EP_1,\cdots,EP_{n_{TF}},EP_{n_{TF}+1},\cdots,EP_{n_{TF}+n_{GY}}\}$		Egalités de puissance.
Concept ⑤ Phénomènes physiques élémentaires	n_{PPE1} PPE1 : $\{PPE1_1,\cdots,PPE1_{n_{PPE1}}\}$		Phénomènes physiques élémentaires 1 port.
	n_{PPEM} PPEM : $\{PPEM_1,\cdots,PPEM_{n_{PPEM}}\}$		Phénomènes physiques élémentaires multiport.
Concept ⑥ Type de phénomène	5 TYPE : {stockageI,stockageC,dissipation, sourceE,sourceF}		Types de phénomènes énergétiques.
Concept ⑦ Lois de comportement	n_{LC} LC : $\{LC_1,\cdots,LC_{n_{LC}}\}$		Lois de comportement.
	$n_{PPE1}+n_{PPEM}$ LCBP : $\{LCBP_1,\cdots,LCBP_{n_{PPE1}},LCBP_{n_{PPE1}+1},\cdots,LCBP_{n_{PPE1}+n_{PPEM}}\}$		Bilans de puissance.
Concept ⑧ Variables de puissance / d'énergie	n_P E : $\{E_1,\cdots,E_{n_P}\}$		Variables d'effort.
	n_P F : $\{F_1,\cdots,F_{n_P}\}$		Variables de flux.
	n_M M : $\{M_1,\cdots,M_{n_M}\}$	$0\leqslant n_M \leqslant n_P$	Variables de moment généralisé.
	n_D D : $\{D_1,\cdots,D_{n_D}\}$	$0\leqslant n_D \leqslant n_P$	Variables de déplacement généralisé.

***Figure 10 –** Diagramme des associations entre les concepts d'un modèle de connaissance.*

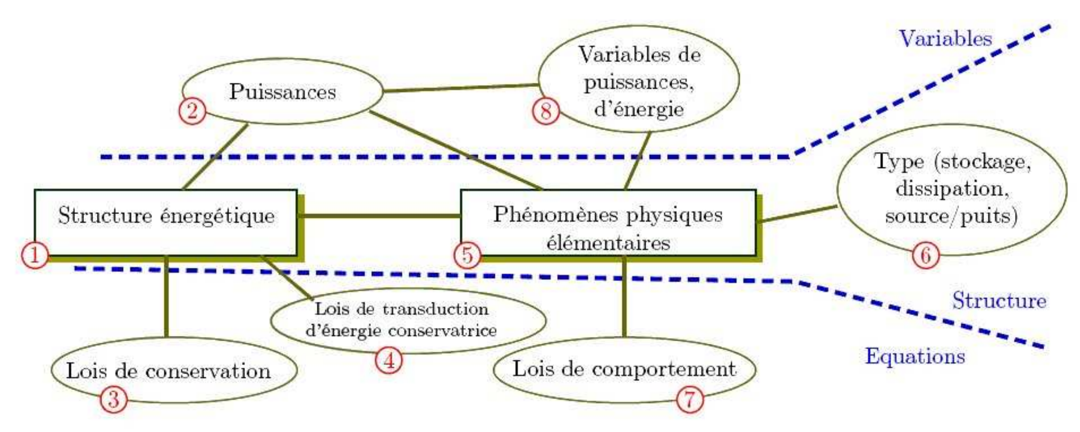


***Figure 11 –** Diagramme des associations entre les concepts d'un modèle de connaissance*

5.3.2 Algorithme d'analyse structurelle

Non supporté pour le moment (en cours de réalisation).

Dans cette méthodologie, la phase première d'analyse structurelle est essentielle pour d'une part, savoir si le problème de conception/dimensionnement est bien posé en terme de cahier des charges/spécifications par rapport au modèles de conception manipulés, et d'autre part, donner les contraintes pour orienter les équations dans le sens inverse requis pour la spécification/sélection/validation des composants en simulation.

Les entrées de l'algorithme ont été, dans la mesure du possible, pensées indépendamment du formalisme d'entrée du modèle. Elles reposent sur les concepts physiques de base présentés précédemment. Les sorties correspondent aux résultats de l'analyse structurelle sur les propriétés structurelles d'un modèle (lignes de puissance, causalité des phénomènes, ordres du modèle, chemin causaux, ordres de ces chemins, ...), sur le bon positionnement du problème entre le cahier des charges et le modèle de conception, et sur les contraintes pour l'orientation des équations en vue de la simulation. L'algorithme proprement dit repose sur les concepts d'analyse structurelle.

Une grande partie du bloc des entrées s'appuie sur les concepts et les associations entre ces concepts décrits de manière générique dans le diagramme de la figure 11.

1. Le premier test consiste à vérifier que le problème posé est carré en termes d'entrées et de sorties. C'est la première condition structurelle pour l'existence du modèle inverse correspondant au problème posé. Si la réponse est négative, soit la procédure s'arrête, soit les entrées doivent être modifiées.
2. L'étape suivante consiste à rechercher les lignes de puissance entrées/sorties.
3. Le test qui suit correspond au critère d'inversibilité structurelle acausale : « Le modèle est structurellement non inversible s'il n'existe aucun choix possible d'ensemble de lignes de puissance entrées/sorties disjointes ». De la même façon, si la réponse est négative, la procédure s'arrête ou les entrées sont à changer.
4. L'étape d'après est relative à l'analyse causale où les chemins causaux entrées/sorties sont déterminés ainsi que leur ordre.
5. Un test analogue au précédent correspond au critère d'inversibilité structurelle causale : « Le modèle est structurellement non inversible s'il n'existe aucun ensemble de chemins causaux entrées/sorties disjoints ».
6. La dernière étape permet le choix de l'ensemble de chemins causaux entrées/sorties disjoints d'ordre minimal et donne les critères de dérivation des sorties.
7. Le test suivant porte sur les classes des sorties spécifiées.

La sortie de l'algorithme, outre les retours d'analyse, porte également les contraintes pour l'orientation des équations en vue de déterminer le modèle inverse.

5.3.3 Résolution des équations

Non supporté pour le moment (en cours de réalisation).

L'objectif est, à partir de l'algorithme et du modèle de données présentés précédemment, de définir les équations ainsi que leur orientation en langage Modelica. Scicos sera utilisé pour trouver les solutions associées au modèle.

Stockage
StockageI
StockageC
BehaviorLaws
Type
Dissipation
Source
SourceE
SourceF
LC
LCBP
ElementaryPhysicalPhenomenon
checkPowerAssociation()
checkType()
MultiPortsPPE
index: int
checkPowerAss...
checkType()
getNumberOfPor...
SinglePortPPE
checkPowerAss...
checkType()
TransductionLaws
EnergyStructure
getPPEMNumber()
Transduction
checkPowerAssociation()
checkEnergyStructureElements()
checkAssociatedTransduction()
checkPPE()
EP
LTE
Variable
Conservation
checkPowerAssociation()
getNumberOfPorts()
checkAssociatedConservation()
checkEnergyStructureElements()
checkPPE()
checkLCE()
GY
checkAssociatedTransduction()
PowerVariable
D
M
TF
checkAssociatedTransduction()
E
getF()
Power
F
getE()
J1
checkAssociatedConservation()
checkLCE()
LCE1
BP
J0
checkAssociatedConservation()
checkLCE()
LCE0
LCB0
LCB1
ConservationLaws

***Figure 12 –** Modèle de donnée UML*

Entrées :
Modèle (phénomènes, structure, lois de comportement, paramètres)
Problème de dimensionnement (sorties spécifiées, entrées à déterminer)

1 Système carré ?
non
Abandon ?
oui
Stop
non
oui
2 Déterminer les lignes de puissance E/S
retour d'analyse
3 Ensemble lignes disjointes ?
non
oui
4 Déterminer les chemins causaux E/S
retour d'analyse
5 Ensemble chemins disjoints ?
non
oui
6 Sélectionner un ensemble d'ordre minimal
Ordres de dérivation des sorties
retour d'analyse
7 Sorties dérivables ?
non
oui

Sorties :
Résultats d'analyse (lignes de puissance, chemins causaux, ordres)
Contraintes d'orientation des équations

Figure 13 – *Algorithme principal lié à l'analyse structurelle*

6 *Réponses à la liste des exigences fonctionnelles*

Dans cette partie nous exposerons les réponses à la liste des exigences fonctionnelles établie par les partenaires industriels IFP, PSA et EDF (cf. document [3]), à laquelle nous avons rajouté les deux exigences 5.1 et 5.2 ci-dessous qui résultent du retour d'expérience sur le compilateur de SimPA (cf. [1] section 2).

6.1 Améliorations liées aux éléments du langage supportés

Le premier point fonctionnel attendu du compilateur Modelica est de supporter un spectre plus large (relativement à SimPA) des éléments du langage Modelica. Cette exigence résulte du retour d'expérience sur SimPA (cf. [1]) dans lequel il a été souligné que les limitations de la syntaxe Modelica supportée par le compilateur ne permettaient pas de traiter des cas de taille industrielle. Suite à ce constat, un certain nombre d'améliorations doivent être apportées au compilateur pour gérer les éléments suivants du langage :

- les tableaux,
- l'héritage,
- les classes de type « package », « record », « model » et « connector » : le compilateur de SimPA ne supportait que « class » (obsolète) et « function » (pour les fonctions externes),
- les attributs « fixed », « nominal », « min » et « max » pour les types prédéfinis,
- les fonctions et opérateurs prédéfinis « pre », « edge », « change », « initial », « terminal », « sample », « reinit », « der », « assert », « terminate », « div », « mod », « rem », « ceil », « floor », « integer » et « smooth »,
- les conversions implicites entre réels et entiers,
- les modifications de classes (et le mot clé « final »),
- les équations conditionnelles,
- les types prédéfinis « Integer » et « Boolean » et les énumérations,
- prise en compte des contraintes de variabilité et discontinuité des expressions et équations,
- la possibilité de définir des connexions hiérarchiques entre modèles (i.e. connexion de sous-modèles),
- possibilité d'appeler des fonctions externes pure écrites en « C » (sans discontinuité ni état interne).

6.2 Gestion des messages d'erreurs dans le compilateur Modelica

Contrairement au compilateur de SimPA qui ne faisait aucun diagnostic des erreurs rencontrées lors de la compilation du code Modelica, le nouveau compilateur affiche des messages d'erreurs clairs. De plus, pour les erreurs syntaxiques et sémantiques des informations précises sont données sur la position de l'erreur dans le fichier source Modelica (ligne et colonne où l'erreur s'est produite), ainsi que des informations complémentaires sur l'origine de l'erreur.

Une liste exhaustive avec une description détaillée des erreurs qui peuvent être rencontrées lors de la compilation est fournie avec la documentation du produit SimPA2.

6.3 Interface homme-machine

6.3.1 Choix des valeurs de départ (variables d'itération) pour l'initialisation

Le compilateur Modelica permet d'initialiser les valeurs des variables en utilisant l'attribut « start ». Sous AMESim il est possible de modifier ou d'attribuer des valeurs initiales pour les variables du modèle en passant par l'interface homme-machine (IHM) standard d'AMESim en mode paramètres.

Dans le code source, il est également possible d'utiliser l'attribut « fixed » pour les paramètres et variables. Ce qui permet :

- d'autoriser ou interdire la modification de certains paramètres
- de fixer ou relâcher les valeurs utilisées pour l'initialisation des variables.

Sous Scicos, les valeurs initiales des variables et des paramètres du modèle peuvent être attribués par l'interface homme-machine (IHM) standard de Scicos en cliquant sur les blocs. Il est, bien entendu, possible d'utiliser l'attribut « fixed » pour les paramètres et les variables. Ce qui permet :

- de remplacer les paramètres fixes par leur valeur numérique au moment de compilation ou générer un code flat. Seuls les paramètres non-fixés seront afficher dans IHM d'initialisation. ces paramètres peuvent être libérés pour être calculé comme des variables

- de fixer ou libérer les valeurs utilisées pour l'initialisation des variables.

6.3.2 Possibilité de choisir le formalisme équationnel pour construire le modèle

Depuis la sortie de la version Rev7 d'AMESim (mai 2007), l'utilisateur a la possibilité d'importer et simuler dans l'environnement d'AMESim des modèles décrits dans un formalisme équationnel en utilisant le langage Modelica.

De plus, un éditeur de code Modelica intégré dans AMESim est prévu dans la version Rev8A (mai 2008).

6.3.3 Possibilité de choisir le formalisme Schémas-Blocs (causal, pour le Contrôle-Commande) pour construire le modèle

Le support du formalisme schéma bloc Modelica est prévu dans AMESim dans le cadre du développement de la librairie Modelica Signal, dont une première version est prévue avec la sortie de la Rev9A (mai 2009).

La construction des diagrammes avec le formalisme schéma bloc est déjà possible dans Scicos. Le formalisme de Scicos permet de mixer dans un même schéma des blocs standards et des blocs Modelica. Ces deux types de blocs peuvent être reliés avec des signaux explicites (entrées/sortie explicites).

6.3.4 Possibilité de choisir le formalisme Bond Graph pour construire le modèle (pas prévu)

Aucun développement court terme n'est prévu pour l'utilisation directe du formalisme Bond-Graph dans AMESim et Scicos.

6.3.5 Possibilité de choisir le formalisme Technologique (assemblage de composants) pour construire le modèle

Une version projet d'assemblage Modelica est prévue dans la version d'AMESim de mai 2008, une version prototype devrait être disponible dés Mars 2008. La première version industrielle est prévue pour mi-2009.

La construction des diagrammes avec l'assemblage des composants est déjà possible dans Scicos. Il existe plusieurs exemples (dans le répertoire démo de Scicos) montrant l'assemblage de composants thermo-hydrauliques ou des composants électriques pour construire des circuits électrique. Dans ces exemples, les liens reliant les ports ne sont pas orientés et les ports sont les ports implicites ce qui signifie il n'existe pas la notion d'entrée/sortie.

6.3.6 Possibilité de rentrer des cartographies ou abaques

Les cartographies peuvent être utilisées en définissant des fonctions externes (écrites en « C »). C'est le seul moyen disponible dans la version SimPA2 du compilateur, car les algorithmes ne sont pas supportés. En dehors du cadre du projet SimPA2, le support des tables d'interpolation fait aussi l'objet d'une étude dans le projet EUROSYSLIB (en cours de spécification). Ceci dit, en pratique l'utilisation des fonctions externes soulève plusieurs problèmes qui font toujours l'objet de discussions et d'études au sein de l'association Modelica. On peut citer en particulier, les problèmes liés à l'utilisation de fonctions impures, et la représentation des discontinuités cachées dans les fonctions externes.

Sous Scicos, un nouveau bloc standard permet de lire les fichiers Excel pour lire les cartographies 2D. Le développement d'un nouveau bloc Excel « n-D » est prévu. Celui-ci pourra être utilisé avec les composants Modelica. Dans Modelica il n'existe pas de dispositif pour lire les fichiers Excel. Pour concevoir un modèle transportable, l'utilisateur devrait développer le modèle soit avec les vecteur, ce qui n'est pratiquement pas possible, soit utiliser les fonctions externes pour construire les cartographies.

6.3.7 Possibilité de fixer et libérer les variables lors de l'initialisation pour le calcul inverse

Sous AMESim, un prototype pour le dimensionnement statique est disponible (depuis mars 2007). Une première version projet est prévue avec la sortie de la Rev9A (mai 2009).

Scicos utilise le compilateur Modelica pour faire de l'initialisation et du dimensionnement statique. L'interface graphique est disponible, la partie numérique est en cours de développement.

Aujourd'hui, la partie de dimensionnement par méthode inverse n'est pas prévue d'être intégrée directement dans Scicos. Une application « standalone » dont les spécifications sont données au paragraphe 5.3 de ce document est prévue. Cette application permettra de générer les équations du système en langage Modelica. Celles-ci seront résolues par Scicos ou AMESim.

Lors de l'exécution de l'algorithme, certaines étapes permettront d'orienter la résolution du système inverse. Pour l'instant, toutes les fonctionnalités disponibles ne sont pas complètement définies.

6.3.8 Accès aux informations sur la structure des équations, sur leur orientation et sur les conditions d'inversibilité du modèle sous l'ensemble des formalismes

Aucun développement prévu dans le cadre de SimPA2.

6.4 Vérification statique (avant simulation)

6.4.1 Vérification de l'inversibilité du modèle par analyse structurelle et fourniture des conditions d'inversibilité

La génération d'un code XML représentant la structure du graphe d'incidence (graphe des dépendances entre variables et équations du modèle) n'est pas prévue dans le cadre de SimPA2. Par contre ce développement est prévu dans AMESim début 2009 dans le cadre d'un autre projet de recherche (PARADE).

Sous Scicos, aucun développement n'est prévu dans le cadre de SimPA2.

Dans la méthodologie développée par l'INSA, la vérification de l'inversibilité du modèle est effectuée en suivant les critères définis ci-dessous :

1. Le système doit être carré
2. Le système doit répondre au critère d'inversibilité structurelle acausale : il doit exister un ensemble de lignes de puissance entrées/sorties disjointes
3. Le système doit répondre au critère d'inversibilité structurelle causale : il doit exister un ensemble de chemins causaux entrées/sorties disjoints

Ces critères conditionnent l'inversibilité structurelle du modèle et représente donc une condition nécessaire. Si elle n'est pas remplie, le modèle n'est pas inversible. Le problème de dimensionnement posé est alors à reconsidérer.

Ces critères seront implémentés dans l'algorithme de l'application « standalone ». Une passerelle vers Scicos est à l'étude.

6.4.2 Détection des problèmes mal posés

Le compilateur Modelica vérifie que le système d'équation final est bien carré, c'est-à-dire, qu'il n'y a pas plus d'inconnues que d'équations. Dans le cas contraire, un message d'erreur est affiché à l'utilisateur. Cependant, dans le cadre de SimPA2, aucun diagnostique d'erreur n'est effectué pour aider l'utilisateur à détecter l'ensemble des modèles à corriger. Ceci pourrait être fait dans le cadre d'une suite au projet SimPA2.

Pour la détection des problèmes mal posés pour la méthodologie par dimensionnement inverse, le critère utilisé est celui de dérivabilité sur les sorties spécifiées du système. Ce critère est associé aux ordres des chemins causaux entrées/sorties qui déterminent le nombre de dérivations nécessaire pour exprimer les entrées à partir des sorties. Il conditionne l'adéquation des sorties spécifiées dans le cahier des charges à la structure du modèle étudié.

Ce critère sera implémenté dans l'algorithme de l'application « standalone ». Une passerelle vers Scicos est à l'étude.

6.4.3 Détection des index élevés

Pas d'implémentation prévue dans le cadre de SimPA2.

6.4.4 Vérification du réseau (cohérence des ports/connexions)

Le compilateur de SimPA sera amélioré en ajoutant :

- Une vérification plus rigoureuse des types des variables à connecter : propriété « flow », type de base, dimensions pour les tableaux…
- Une vérification des contraintes de connexion : « input » et « output »
- Une extension aux connexions hiérarchiques entre modèles.

En plus des vérifications effectuées par le compilateur, le produit commercial AMESim doit tenir compte des contraintes de connexion entre modèles mixtes (composés de modèles traditionnels AMESim et de modèles Modelica).

6.4.5 Détection des Boucles Algébriques dans le process physique, le contrôle-Commande, et entre le Contrôle Commande et le process physique

Dans AMESim comme dans Scicos, il ne sera pas possible d'afficher les boucles algébriques apparaissant à l'intérieur d'un modèle Modelica. En effet, les boucles algébriques détectées par le compilateur portent sur l'ensemble des variables obtenues après simplification (qui peut être très différent de l'ensemble des variables initiales introduites par l'utilisateur). L'affichage de boucles algébriques sur ces variables n'aura aucun intérêt pour l'utilisateur, car celles-ci peuvent être complètement inconnues de lui. Pour résoudre ce problème, une étude doit être menée sur la possibilité de remonter l'information sur les boucles algébriques vers les variables et équations du modèle initiale (avant simplification). Cette étude ne peut pas se faire dans le cadre du projet SimPA2.

D'un autre côté, dans AMESim 8A il sera possible d'afficher les boucles algébriques générées par des assemblages de sous modèles AMESim. En d'autres termes, il ne sera pas possible d'afficher les boucles algébriques rencontrées à l'intérieur d'un même modèle Modelica, mais il sera possible d'afficher les boucles qui apparaissent entre plusieurs sous modèles AMESim (standards ou importés à partir d'un code Modelica).

6.5 Génération de modèle

6.5.1 Informations sur le système (nombre d'équations, de paramètres, d'états, de blocs,…)

Déjà supporté dans AMESim et Scicos.

6.5.2 Génération de code temps réel embarqué à partir d'un contrôleur discret

Sous Scicos, aucun développement n'est prévu dans le cadre de SimPA2.

6.6 Initialisation

6.6.1 Calcul Inverse (possibilité de définir et résoudre l'état initial comme un problème inverse)

Cette fonctionnalité est couverte par le compilateur Modelica, puisqu'il se base sur une description acausale du système à étudier à travers le langage Modelica. Ainsi les valeurs des paramètres et variables à l'état initial peuvent être tout naturellement considérées comme des inconnues du système d'équations à résoudre.

6.6.2 Résolution efficace des Boucles Algébriques (résolution de l'état initial comme état stationnaire)

Cette fonctionnalité est implantée sous AMESim depuis la sortie de la Rev7 (mai 2007) et existe sous Scicos depuis le projet SimPA.

Les algorithmes de résolution statiques doivent être améliorés dans AMESim et Scicos.

Des travaux ont été entamés en 2007 dans Scicos. L'interface graphique d'initialisation permet d'initialiser les modèles en état stationnaire, c'est-à-dire avec les dérivées nulles.

Des travaux pour améliorer les calculs d'initialisation dans AMESim sont planifiés en 2008 (amélioration de l'existant et exploration de nouvelles méthodes).

6.6.3 Résolution des discontinuités (commutation de modèles). Résolution du problème de la détermination des bons modes au démarrage de la simulation.

Pour permettre la description de modèles pouvant présenter suivant leur point de fonctionnement des structures d'équations différentes, la syntaxe supportée par le compilateur Modelica a été étendue pour pouvoir utiliser les « équations conditionnelles » de la forme :

if condition1 **then** equation1 **elseif** condition2 **then** equation 2 **else** equation 3 **end if**;

6.7 Simulation

6.7.1 Calcul direct

Cette fonctionnalité est supportée :
- par AMESim depuis la version REV7 (mai 2007),
- par Scicos.

6.7.2 Traitement des discontinuités (commutation de modèles)

Cette fonctionnalité est déjà supportée par Scicos. Elle sera supportée par AMESim avec la sortie de la version Rev8A (mai 2008).

6.7.3 Traitement des boucles Algébriques (DAE)

Cette fonctionnalité est supportée dans les deux environnements AMESim et Scicos.

6.7.4 Traitement des systèmes Raides (DAE et ODE)

Cette fonctionnalité est supportée dans les deux environnements AMESim et Scicos.

6.7.5 Traitement des systèmes hybrides (coexistence de modèles continus et discrets)

Prévu courant 2008 dans Scicos puis ensuite dans AMESim. Les travaux menés depuis mai 2007 ont permis de résoudre un certain nombre de problèmes liés aux systèmes discrets dans Modelica (définition d'un modèle rigoureux permettant des simulations performantes).

6.7.6 Calcul inverse automatisé en cours de simulation

Aucun développement prévu sous AMESim ou Scicos pour supporter cette fonctionnalité.

6.7.7 Couplage de modèles 1D et 3D

Aucun développement prévu sous AMESim pour supporter cette fonctionnalité. Pour la partie Scicos, une application externe permettra de générer le système d'équation inverse en langage Modelica. Ce système pourra ensuite être résolu par Scicos ou AMESim.

6.8 Outils de diagnostic

6.8.1 Informations sur la structure des équations effectivement résolues (après manipulations symboliques) données sous forme littérale compréhensible par l'utilisateur (liste des systèmes d'équations après réduction, …)

Non supporté (manque de spécification claire).

6.8.2 Accès aux informations sur la structure des équations, leur orientation et les conditions d'inversibilité du modèle (version textuelle)

Sous AMESim : aucun développement prévu pour supporter cette fonctionnalité.

Sous Scicos : une application externe permettra de générer le système d'équation inverse en langage Modelica. Ce système, contenant les équations du modèle ainsi que leur orientation, sera ensuite résolu par Scicos.

6.8.3 Messages d'erreurs en cours de simulation

AMESim supporte depuis sa création un ensemble de fonctionnalités permettant d'analyser le système résolu numériquement.

Scicos se sert du solveur numérique « Sundials ». Ce solveur fournit des messages d'erreur au cours de simulation. Ces messages d'erreurs ainsi que les messages d'erreurs liés à la compilation du modèle sont affichées sur la fenêtre de Scilab.

6.9 Environnement informatique

6.9.1 Plateformes informatiques supportés

Les composants informatiques développés dans le cadre de SimPA2 pourront fonctionner sous toute plate-forme possédant des compilateurs C et Objective Caml récents.

6.9.2 Mise à disposition de la documentation utilisateur

Sous AMESim, la documentation utilisateur de l'interface Modelica est fournie sous la forme d'un document PDF, accessible à partir du menu : « Help/Online/Manuals/Modelica Interface ».

Une aide est également disponible en utilisant le compilateur Modelica en ligne de commande.

6.10 Les performances

L'utilisation du compilateur Modelica devrait permettre d'aborder les « gros modèles » de taille industrielle, et d'améliorer significativement les temps de simulation. En effet, les méthodes de simplification formelle appliquées par le compilateur réduisent considérablement la taille des systèmes effectivement résolus par le solveur et diminuent ainsi les temps de calcul.

6.11 La portabilité des modèles entre AMESim et Scicos

Dans un premier temps, AMESim et Scicos utilisent le même compilateur Modelica, ce qui garantit que les librairies Modelica écrites pour Scicos, peuvent être directement utilisées (sans aucune modification) sous AMESim et vice versa.

Par la suite, le compilateur Modelica sera amélioré (par AMESim) pour traiter d'autres classes de problèmes étendues (réduction d'index, algorithmes, fonctions externes avec discontinuités et états

internes, partitionnement…) dans le cadre d'une version commerciale du compilateur. Dans cette dernière version, la portabilité de Scicos vers AMESim sera maintenue, mais pas l'inverse. C'est-à-dire que certains modèles écrits sous AMESim peuvent ne pas fonctionner sous Scicos.

6.12 La documentation

La documentation fournie avec le compilateur Modelica comportera la description :

- des fonctionnalités du compilateur
- des erreurs syntaxiques et sémantiques
- des éléments non supportés du langage Modelica
- de la procédure d'installation du compilateur.

7 Récapitulatif des réalisations

7.1 Liste des principales fonctionnalités et limitations du compilateur Modelica

Case vide : non supporté, L : supporté avec des limitations, X : complètement supporté				
Fonctionnalité	Compilateur SimPA	Compilateur SimPA2	Version commerciale	
			mai 2008	mai 2009
Fonctions et opérateurs prédéfinis	L	L	X	X
Types prédéfinis	L	L	L	X
Utilisation de packages, records, blocks et connecteurs		X	X	X
Héritage		X	X	X
Tableaux statiques		X	X	X
Tableaux dynamiques				X
Equations conditionnelles		X	X	X
Composants remplaçables				X
Equations initiales				X
Fonctions externes	L	L	L	X
Algorithmes				L
Gestion des messages d'erreurs		L	L	X
Annotations		L	L	L
Support des aspects discrets		L	L	X

Pour le compilateur SimPA2, les limitations (indiquées par « L » dans le tableau ci dessus) sont :

- Certains opérateurs prédéfinis ne sont pas supportés : assert, terminate, initial, terminal, change, edge, les opérateurs de manipulation des tableaux,
- Seul les types « Real » et « Integer » sont supportés (les types « Boolean », « Enumeration » et « String » ne sont pas supportés),
- Les fonctions externes supportées sont celles :
  - ayant au maximum un seul « output »,
  - pour lesquelles tous les « inputs » et « outputs » sont de type « Real »,
  - qui sont pures et sans aucune discontinuité.
- L'affichage des messages d'erreurs est géré et documenté seulement pour la partie front-end du compilateur,
- Les erreurs syntaxiques sur les annotations sont bien supportées par le compilateur, mais aucune information sur les annotations n'est générée. C'est à l'outil de simulation de les récupérer et les interpréter.
- Concernant les aspects discrets, les éléments supportés sont :
  - les réinitialisations d'états continus,
  - les états discrets indépendants de type Real (« indépendants » signifie qu'on ne gère pas les cascades d'événements provoquées par des «when» dépendants).

Les éléments précédents concernent le compilateur SimPA2 disponible en mai 2008. Une prochaine étape (mai 2009) sera l'intégration du travail conjoint INRIA-Imagine (intégration du nouveau modèle hybride).

7.2 Intégration de Modelica dans AMESim

Certains des projets liés à l'intégration de Modelica dans AMESim sont livrés en trois étapes:

- version prototype : qui est une première version, destinée aux tests en interne,
- version projet : qui est réservé à deux ou trois clients pour tester le produit et avoir les premiers retours des utilisateurs,
- version industrielle : qui est la version finale destinée à tous les clients.

Date	Version	Fonctionnalités
mai 2007	Rev7	- Import Modelica dans AMESim (simulation directe), - Dimensionnement statique (version prototype), - Génération de messages d'erreurs explicites au cours de la compilation
mai 2008	Rev8A	- Import Modelica amélioré - Editeur de code Modelica (avec compilateur intégré), - Assemblage Modelica (version projet).
mai 2009	Rev9A	- Dimensionnement statique (version projet) - Assemblage Modelica (version industrielle)

7.3 Intégration de Modelica dans Scicos

Date	Version	Fonctionnalités
juin 2007	4.1	- Le simulateur stand-alone Scicos, - Editeur direct de code Modelica, - Intégration du front-end du compilateur dans Scicos, - Affichage des messages d'erreurs de la compilation, - Intégration de Sundials dans Scicos
mai 2008	4.2	- Prototype pour le dimensionnement statique dans Scicos, - Intégration de nouveaux solveurs/méthodes d'initialisation dans Scicos (prototype).
mai 2009	4.3 ou 5	- Dimensionnement statique, Initialisation des modèles Modelica.

8 Références

[1] Projet SimPA2 - Expression de besoins pour un environnement de modélisation et simulation basé sur un compilateur Modelica open source, aout 2006.

[2] RNTL – Appel à propositions 2005, Annexe technique du projet SimPA2.

[3] Définition du tableau des exigences fonctionnelles pour l'établissement de la roadmap SimPA2, février 2007.

[4] Accord de consortium du projet SimPA2.

[5] Modelica – A unified object-oriented language for physical systems modeling, Language specification version 3.0, Sept. 5, 2007.

[6] S. Furic (2007) Software design specification for SimPA2 – SP2.

[7] P. Fritzson (2004) Principles of object oriented modeling and simulation with Modelica 2.1, Wiley-IEEE Press.

[8] M. Najafi et R.Nikoukhah (2008) Initialization of Modelica models in Scicos, to appear in 6th International Modelica Conference.